Pasi Österman

RFID-TUNNISTUS- JA SEURANTAJÄRJESTELMÄ

Insinööri(AMK)
Tietotekniikan koulutusohjelma
2011

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

RFID TUNNISTUS JA SEURANTAJÄRJESTELMÄ

Österman, Pasi
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Syyskuu 2011
Ohjaajat: Javanainen, Mikko ja Kompuinen, Ritva
Sivumäärä: 42
Liitteitä: 3

Asiasanat: RFID, EPC, tunnistus, seuranta, ohjelmointi, tietokanta, tunniste

Tämän opinnäytetyön aiheena on RFID ohjelmistokehitys. RFID-opinnot ovat jo jonkin aikaa olleet osa Satakunnan ammattikorkeakoulun tarjoamaa opetusta. Opetuksessa syntyi tarve käytännölliselle RFID-sovellukselle. Tämän opinnäytetyön tarkoituksena oli tutkia RFID-tekniikkaa ja yleisiä käyttökohteita, sekä tuottaa koulun tarpeita vastaava RFID-sovellus.

RFID-seuranta- ja tunnistusjärjestelmä osoittautui opetuksen kannalta parhaimmaksi vaihtoehdoksi. Ohjelma suunniteltiin käyttämään kolmen tason kerrosarkkitehtuuria ja tehdasmetodisuunnittelumallia. Näiden ratkaisujen tavoitteena oli helpottaa ohjelman jatkokehitystä. Tunnisteiden tunnistenumeron paikalla päädyin käyttämään sähköistä tuotekoodia parantaakseni järjestelmän pitkäikäisyyttä.

Opinnäytetyön kautta koulu sai käyttöönsä RFID-sovelluksen, mitä voidaan käyttää RFID-opetuksessa tulevaisuudessa.

RFID TAG IDENTIFICATION AND TRACKING SYSTEM

Österman, Pasi
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Information technology
September 2011
Supervisors: Javanainen, Mikko ja Kompuinen, Ritva
Number of pages: 42
Appendices: 3

Keywords: RFID, EPC, identification, tracking, programming, database, tag

The subject of this thesis is RFID software development. Faculty of technology and maritime management Rauma campus of Satakunta University of applied sciences has had RFID as part of their education for some time now. However the lack of practical software application for the schools RFID system has become a problem. Purpose of this thesis was to recearch RFID technology and how it's commonly used and to provide the school with fitting RFID application.

In conclusion to the research the most fitting application for schools RFID lessons was a RFID application that would use RFID tags to identify and keep track of items. I designed the software to use three-tier software architecture and use the common factory method design pattern. For Tag identification purposes the application uses Access database to identify and track the RFID Tags encoded with EPC SGTIN-96 scheme.

SISÄLLYS

1 JOHDANTO

1.1 Opinnäytetyön tarkoitus.

Radiotaajuisen etätunnistamisen yleistyessä, myös tähän liittyvän koulutuksen tarve kasvaa. Satakunnan ammattikorkeakoulun Tekniikan ja merenkulun Rauman toimipisteessä on jonkin aikaa löytynyt tarvetta toimivalle RFID-sovellukselle, joka vastaa tekniikan käyttöä yritysmaailmassa ja täyttää opetuksen tarpeet. Koulun nykyinen ohjelmisto on tältä osin puutteellinen ja tarkoitettu lähinnä RFID-järjestelmän komponenttien testaukseen.

Tämän päättötyön tavoitteena on toteuttaa ohjelmistoprojektina koulun RFID-järjestelmälle opetukseen soveltuva RFID-sovellus, joka vastaa tekniikan nykyisiä käyttökohteita. Ohjelma toteutetaan Windows-käyttöjärjestelmälle käyttäen C# ohjelmointikieltä.

1.2 Tavoitteet.

Opinnäytetyön tavoitteena on tutustua perusteellisesti RFID-tekniikkaan, ohjelmistoprojektin vaiheisiin sekä käytössä oleviin menetelmiin. Koska päättötyö tehdään Satakunnan Ammattikorkeakoulun opetuskäyttöön, jatkokehityksen kannalta on tärkeää, että ohjelma on helposti ylläpidettävissä ja toiminnallisuus on selkeästi dokumentoitu.

1.3 Satakunnan ammattikorkeakoulu.

Satakunnan ammattikorkeakoulussa opiskelee noin 6500 opiskelija ja se työllistää noin 500 asiantuntijaa. Koulutusta tarjotaan Porissa, Raumalla, Huittisissa ja Kankaanpäässä. Toimialoja ovat mm. liiketoiminta ja kulttuuri, sosiaali- ja terveysala, sekä tekniikka ja merenkulku.

2 YLEISTÄ

2.1 Ohjelmistoprojektin aloittaminen

Monille aloitteleville ohjelmoijille ei ole välttämättä heti selvää, että ohjelmointiprojektin aloittaminen ei ole niin yksinkertaista, miltä se saattaa kuulostaa. Aloitettaessa suurin vastaantuleva kysymys on, mistä oikein kannattaa aloittaa ja miten?

Ohjelmistoprojektin lähtöasetelma osoittautui oletettua haastavammaksi. Useimmissa ohjelmistoprojekteissa asiakkaalla on tarkka käsitys siitä, minkälaisen ohjelman hän tarvitsee, mutta tässä näin ei ollut. Ensimmäiseksi tehtäväkseni osoittautui siis selvittää RFID-opetuksen tarpeet ja tämän perusteella ideoida minkälainen käytännön sovellus näitä parhaiten palvelisi. Ongelmaa helpotti aikaisemmin käymäni kurssi aiheesta, josta sain kattavan kuvan aiheen nykyisestä opetuksesta ja sen kehittämiskohteista.

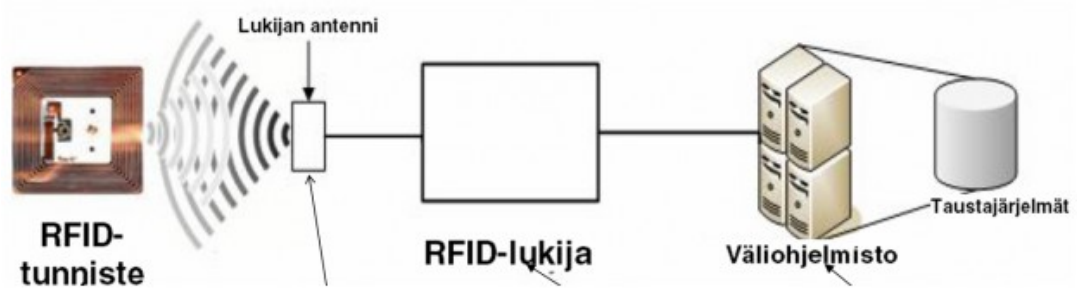
Ongelmien ratkaisemiseksi katsoin tarpeelliseksi perehtyä syvemmin radiotaajuiseen etätunnistukseen sekä tämän tekniikan yleisiin käyttökohteisiin.

2.2 Radiotaajuinen etätunnistus

Radiotaajuisessa etätunnistuksessa (RFID, Radio Frequency Identification) on pohjimmiltaan kysymys esineiden ja asioiden tunnistamisesta radioaaltojen avulla. Tekniikkaa on helppo ajatella viivakoodin seuraajana, sillä periaate on samankaltainen. Tunnistettavaan esineeseen kiinnitetään tunniste, johon on tallennettu oma sarjanumero. Tämä sarjanumero luetaan radioaaltojen avulla lukijalaitteella. Viivakoodista poiketen lukijalaitteen ei tarvitse olla näköyhteydessä tunnisteen kanssa ja tunnisteen sarjanumeroa voidaan halutessa muokata. (RFID journal www-sivut n.d; RFIDLab www-sivut n.d)

2.2.1 RFID-järjestelmä

RFID-järjestelmään kuuluu tyypillisesti tunnisteet, lukijalaite antenneineen, väliohjelmisto. Väliohjelmisto voi olla yhteydessä yhteen tai useampaan taustajärjestelmään, esimerkiksi tietokantaan. Tunniste on käytännössä mikrosiru, johon on tallennettu tunnistetiedot. Lukijalaitteen tehtävä järjestelmässä on kommunikoida ympäristössä olevien tunnisteiden kanssa. Tämän se tekee siihen kiinnitettyjen antenniensa avulla.



Kuva 1. RFID-järjestelmän komponentit (RFIDLab www-sivut n.d)

Järjestelmän toimintaperiaate voisi ohjelmiston kehittäjän kannalta olla seuraavanlainen. Lukijalaitetta hallitseva ohjelma lähettää laitteelle lukupyynnön, minkä seurauksena lukijalaite etsii ympäristöstä tunnisteita. Tämä tapahtuu lähettämällä antenneilla radiotaajuista signaalia ympäristöön, johon tunnisteet vastaavat omilla tunnistetiedoillaan. Lukijalaite vastaanottaa tunnistetiedot ja ohjaa ne niitä pyytäneelle ohjelmalle, joka käsittelee saamansa tiedot haluamallaan tavalla. (RFIDLab www-sivut 2011; Glover & Bhatt 2006, 36)

2.2.2 Tunnisteiden kotelointi

Tunnisteet on mahdollista koteloida useisiin eri käyttötarkoituksiin. Tuotteisiin kiinnitettävät tarrat, mukana kulkevat avaimenperät ja henkilökortti ovat vain muutamia esimerkkejä mahdollisesta koteloinnista. Kotelointi kannattaa tietysti valita käyttötarkoituksen mukaan, esimerkiksi vaikeissa säilytys tai kuljetusolosuhteissa muovikoteloitu tunniste saattaa osoittautua paremmaksi vaihtoehdoksi kuin tunnistetarra.
(Glover & Bhatt 2006, 35)

2.2.3 Aktiiviset, passiiviset ja puolipassiiviset tunnisteet

Se, onko tunniste aktiivinen, passiivinen vai puolipassiivinen määrittyy sen mukaan onko tunnisteessa oma akku. Aktiivisissa tunnisteissa mikrosiru saa virtansa akulta, kun passiivisissa tunnisteissa mikrosiru saa virtansa lukijalaitteelta. Puolipassiivinen tunniste sen sijaan käyttää akkuaan mikrosirun virranlähteenä, mutta kommunikoi lukijalaitteelta saamallaan virralla. Aktiivisten ja puolipassiivisten tunnisteiden etu passiivisiin tunnisteisiin nähden on pidempi lukuetäisyys, mutta vastaavasti ne myös maksavat huomattavasti enemmän. (Sarekoski www-sivut. n.d.)

2.2.4 Gen2 tunnisteet

Gen2 tunnisteet ovat nimensä mukaisesti toisen sukupolven RFID-tunnisteita. Tunniteet toimivat EPCglobalin laatiman Gen2-standardin mukaisesti. Standardin mukaan tunnisteet voidaan lukea miltä tahansa taajuudelta taajuusalla 860-960MHz. (SkyRFID www-sivut n.d)

2.2.5 Gen2 tunnisteiden muisti

Gen2-tunnisteiden muisti on jaettu neljään muistilohkoon. Neljäs muistilohko on käytännössä ylimääräinen, eikä sitä löydy kaikista tunnisteista. Tunnisteiden tietoturvasta vastaa erillinen salasanoille tarkoitettu lohko, jolla voidaan estää asiattomien pääsy tunnisteen tietosisältöön. Lisäksi voidaan määrittää erillinen salasana myös tunnisteiden tuhoamiseen.

Taulukko 1. Tunnisteen muistilohkot (UHF Cass 1 Gen2 Standard v. 1.2.0, 2008, 37)

EPC memory (Tag ID)	Tarkoitettu tunnistenumerolle kuten EPC. Numero kertoo yleensä mihin tunniste on kiinnitetty.
TID memory	Sisältää tunniste tai valmistajakohtaista tietoa.
Reserved memory	Sisältää tunnisteen tuhoamiseen ja käyttöön tarvittavat salasanat.
User memory	Lisämuisti mitä voi käyttää vapaasti omiin tarpeisiin.

Tunnisteiden tietokapasiteetti saattaa parhaimmillaan vaihdella muutamasta bitistä tuhansiin bitteihin. Koska valtavien binäärilukujen käsittely on yleisesti hankalaa, käsitellään tunnisteen muistilohkoja yleensä heksadesimaalilukuina. Ohjelman kannalta

tärkein muistilohko kuuluu tunnistenumerolle, jonka pituus on 24 heksamerkkiä (96-bittiä).

2.2.6 EPC, sähköinen tuotekoodi

Sähköinen tuotekoodi (EPC, Electronic Product Code) on EPCglobalin hallinnoiva globaali numerointistandardi tunnisteiden tietosisällöille. Standardin tavoitteena on tehdä RFID-sovelluksista maailmanlaajuisesti yhteensopivampia. EPCglobalin visiona on mahdollistaa sähköisellä tuotekoodilla tuotteiden maailmanlaajuinen seuranta EPC-tietojärjestelmän avulla.

Sähköinen tuotekoodi on yleensä noin 24 merkkiä pitkä heksadesimaaliluku, joka tallennetaan EPC memory muistilohkoon. Viivakoodin tapaan sähköinenkään tuotekoodi ei sellaisenaan kerro käyttäjälle paljoa sen tietosisällöstä. Tämän takia sähköinen tuotekoodi on erikseen käännettävä helpommin ymmärrettävään muotoon.
(Wikipedia www-sivut n.d; RFIDLab www-sivut 2011)

2.3 Käyttökohteet

Saadessani käsityksen tekniikasta ja siihen läheisesti liittyvistä standardeista jatkoin tutkimustyötä perehtymällä tekniikan eri käyttökohteisiin. Koska tarkoituksenani on tehdä ohjelmisto opetuskäyttöön, on sen vastattava RFID-tekniikan nykyisiä käyttökohteita. Koulumme tarjoaa insinööriopetusta mm. logistiikkaan, kone- ja tuotantotekniikkaan ja tuotantotalouteen. Siksi katsoin parhaaksi keskittyä juuri näiden alojen kannalta oleellisiin radiotaajuisen etätunnistuksen käyttökohteisiin.

2.3.1 RFID logistiikassa

Radiotaajuista etätunnistamista on ruvettu hyödyntämään useissa eri logistisissa sovelluksissa. Tunnisteiden avulla on helppo seurata tuotteiden liikettä reaaliajassa ja kerätä tämän perusteella arvokasta lisätietoa. Tietojen perusteella on esimerkiksi mahdollista paikantaa tuotteiden nykyinen sijainti. RFID-sovellusten on todettu vähentävän hävikin määrää ja nopeuttavan tuotteiden käsittelyä.

Radioaallot ja sarjanumerot mahdollistavat myös yksittäisten tuotteiden havaitsemisen suuremmasta kokonaisuudesta, esimerkiksi pahvilaatikon sisältö voidaan saada selville ilman laatikon avaamista. Ominaisuuden hyödyntäminen asettaa rajoitteita pakkausmateriaalille, sillä radioaallot kimpoavat metallista. (S.Sarekoski www-sivut. n.d; Granqvist, Permala, Scholliers. 2007)

2.3.2 RFID tuotannossa

Tuotannossa RFID-tekniikalle löytyy useita käyttökohteita. Teknologia mahdollistaa esimerkiksi älykkäät tuotantolinjat, jossa voidaan samanaikaisesti valmistaa useita eri tuotteita. Tuotantolinja tunnistaa tuotteen yksilönä ja tekee sille ennalta määritellyt toimenpiteet.

Tunnisteet mahdollistavat myös tuotteiden tarkan seurannan tuotantolinjassa, jonka vuoksi tuotteiden työstöjärjestystä voidaan optimoida ja täten minimoida mahdolliset jonotusajat. Tätä seurantaa on mahdollista hyödyntää myös laadunvalvonnassa, jossa tuotteen tietoihin voidaan heti tallentaa mahdolliset laatupoikkeamat sekä tarvittavat

menettelyt. Tietyissä tilanteissa on myös mahdollista paikantaa ja pysäyttää linjaston osa, jossa laatupoikkeama tapahtui. Näin vältytään laitevahingoilta ja mahdollisesti vähennetään hävikin määrää. (Salvén, www-sivut, 2011)

2.3.3 RFID kaupoissa

Tulevaisuuden kaupoissa tuotteiden hintatiedot voitaisiin lukea suoraan ostoskärryistä. Kauppojen hyllyjärjestelmät ilmoittaisivat heti tapahtumista, kuten tuotteiden loppumisesta, päiväysten vanhenemisesta tai tuotteen väärästä sijainnista. Samalla tapaa myös varastojärjestelmät voisivat ylläpitää tietoja tuotteistaan.

2.4 Koulun RFID-järjestelmä

Tutkimustyössä perehdyin myös koulun nykyiseen RFID-järjestelmään ja ohjelmistoon. Näin sain paremman käsityksen nykyisen ohjelmiston puutteista ja siitä millaisen ohjelmiston toteuttaminen laitteelle olisi ylipäätään mahdollista.

2.4.1 Lukijalaite

Koulun lukijalaitteena toimii Siritin valmistama Infinity 510 malli. Laite koostuu lukijalaitteesta, johon on kaapeleilla kiinnitetty neljä antennia. Lukijalaite on kiinnitetty eräänlaiseen porttiin ja antennit sijoitettu molemmin puolin portin sisäpuolelle.

Kuva 2. Koulun RFID-lukijalaite: Sirit Infinity 510

Ensitesteissä lukijalaite toimi jopa liiankin hyvin, sillä se tunnisti portin sisällä olevien tunnisteiden lisäksi myös pöydällä olevat tunnisteet. Lukuetäisyyttä oli siis rajoitettava antennien tehokkuutta laskemalla.

Koulun lukijalaite on kiinnitetty sitä käyttävään tietokoneeseen kytkimen kautta perus RJ-45 kaapelilla. Lukijalaitteen tapa kommunikoida tietokoneen kanssa mahdollistaa useiden lukijalaitteiden käytön yhdeltä tietokoneelta.

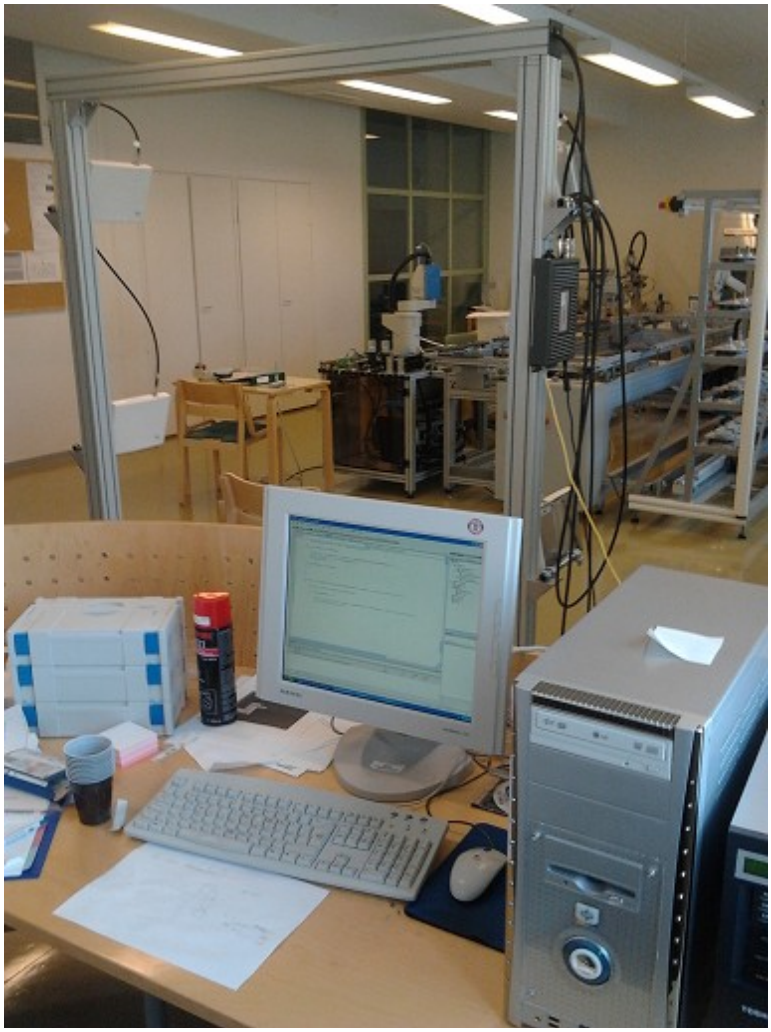
2.4.2 Tunnisteet

Koulun opetuskäytössä on kahden tyyppisiä RFID-tunnisteita. Koulutuksessa yleisimmässä käytössä ovat tarratunnisteet, joita tulostetaan erillisellä Toshiban RFID-yhteensopivalla tarratulostimella. Tarratunnisteet ovat tyypiltään 96-bittisiä Gen2-

tunnisteita. Tarratunnisteiden lisäksi koululta löytyy myös muovikoteloituja 96-bittisiä Gen2-tunnisteita, joista löytyy myös käyttäjälle tarkoitettu user memory muistilohko.

2.4.3 Tietokone

Lukijalaitetta ohjataan muutaman vuoden ikäisellä Windows XP -käyttöjärjestelmällä varustetulla koneella. Koneeseen on asennettu RFID -testiohjelmiston lisäksi myös monia muita opetuksessa käytettäviä ohjelmistoja. Näistä ohjelmointiprojektin kannalta tärkeimpiä ovat Visual Studio 2008 ja Microsoft Office Access.



Kuva 4. Koulun RFID-järjestelmä.

3 MÄÄRITTELY

Ohjelmiston määrittelyn tarkoituksena ohjelmistoprojektissa on määrittää, minkälainen ohjelma projektin lopputuloksena oikein syntyy. Ohjelmiston määrittelyssä kuvataan se ohjelma, joka täyttää asiakkaan tarpeet ja sille asetetut vaatimukset. Määrittelyn tuloksena syntyy toiminnallinen määrittelydokumentti, johon on kerätty sekä vaatimukset, että vaatimuksen täyttävän ohjelmiston kuvaus. (Haikala & Märijärvi, 2004, 78)

Kirjoitin ohjelmalle toiminnallisen määrittelyn asiakkaan kanssa käytyjen keskusteluiden sekä oman käsitykseni mukaan. Kyseisen dokumentin pohjana käytin HYTT laatujärjestelmästä löytyvää teknisen määrittelyn pohjaa. Tämän määrittelydokumentin sekä ohjelmiston varhaisten prototyyppien avulla pääsimme asiakkaan kanssa yhteisymmärrykseen valmistuvasta ohjelmasta. Tässä opinnäytetyön luvussa on mielestäni opinnäytetyön kannalta oleellisimmat asiat toiminnallisesta määrittelystä.

3.1 Järjestelmän kuvaus ja vaatimukset

Saadessani paremman käsityksen radiotaajuisesta etätunnistuksesta helpottui myös vaatimusten määrittely. Standardit ja tekniikan käyttökohteet antoivat paljon lisätietoa asioista mihin sovellusta tehdessä kannattaa kiinnittää huomiota.

Asiakkaan alkuperäinen idea opinnäytetyölle oli toteuttaa ohjelma, joka hakee ja tallentaa tunnisteisiin liittyvää tietoa tietokantaan. Pohdittuani jonkin aikaa mahdollisia ideanmukaisia sovelluksia päädyimme siihen tulokseen, että ideaa vastasi parhaiten RFID-tunnistus- ja seurantajärjestelmä. Eli tarvittiin yksinkertainen järjestelmä, joka lukee tunnisteet lukijalaitteen avulla ja hakee tunnistenumeron avulla tarvittavat lisätiedot.

Tärkeimmät vaatimukset olikin tämän jälkeen huomattavasti helpompi listata.

- Järjestelmän on pystyttävä käsittelemään useita tunnisteita.
- Tunnisteiden tietoja pitää pystyä muokkaamaan.
- Ohjelma pitää olla helposti päivitettävissä.
- Ohjelma ei saa olla käyttäjälle liian monimutkainen.
- Ohjelman pitää toimia nykyisellä laitteistolla.
- Ohjelmakoodi pitää olla hyvin kommentoitu.

3.2 Toiminnallisuus

3.2.1 Tunnisteiden lukeminen

Ohjelma aloittaa tunnisteiden lukemiseen tarkoitetun prosessin käyttäjän painaessa Aloita luenta -painiketta. Lukuprosessin aikana ohjelma listaa kaikki lukijalaitteen havaitsemat tunnisteet käyttäjälle. Lukuprosessi pysyy käynnissä niin kauan, kunnes käyttäjä painaa Lopeta luenta -painiketta. Tunnisteita listatessa ohjelma listaa jokaisen tunnisteen vain kerran. Listan voi tyhjentää painamalla Listan tyhjennys -painiketta.

3.2.2 Tunnisteiden lisääminen

Käyttäjä voi lisätä ohjelman kautta tietokantaan myös uusia tunnisteita. Ohjelma tallentaa tietokantaan uuden tunnisteen käyttäjän syöttämien tietojen perusteella ja muokkaa tunnisteen tunnistetiedot tietojen mukaisiksi. Ohjelma katsoo myös, että jokainen tunniste saa oman yksilöllisen sarjanumeron.

3.2.3 Tunnistetietojen muokkaaminen

Ohjelman kautta voi myös muokata tunnisteiden tietoja. Käyttäjä syöttää tunnisteelle haluamansa muutokset, jonka jälkeen ohjelma tarkistaa ja toteuttaa muutokset sekä tietokantaan että tunnisteeseen.

3.2.4 Tunnisteiden seuranta

Aina ohjelman lukuprosessin ollessa käynnissä, ohjelma tallentaa tietokantaan tunnisteiden viimeisimmän sijainnin ja lukuhetken. Tunnisteiden seuranta tapahtuu siis leimaus, periaatteella ja vastaa kysymyksiin missä ja milloin.

3.3 Tiedot ja tietokannat

3.3.1 RFID-tunnisteen tietosisältö

Tunnisteiden tunnistenumeroiden kohdalla tullaan käyttämään sähköistä tuotekoodia EPCglobalin kehittämän Tag data standard 1.6 version mukaisesti.

Tunnisteiden tunnistenumero määritetään siis alustavasti standardista löytyvän SGTIN-96 koodaustaulukon mukaisesti.

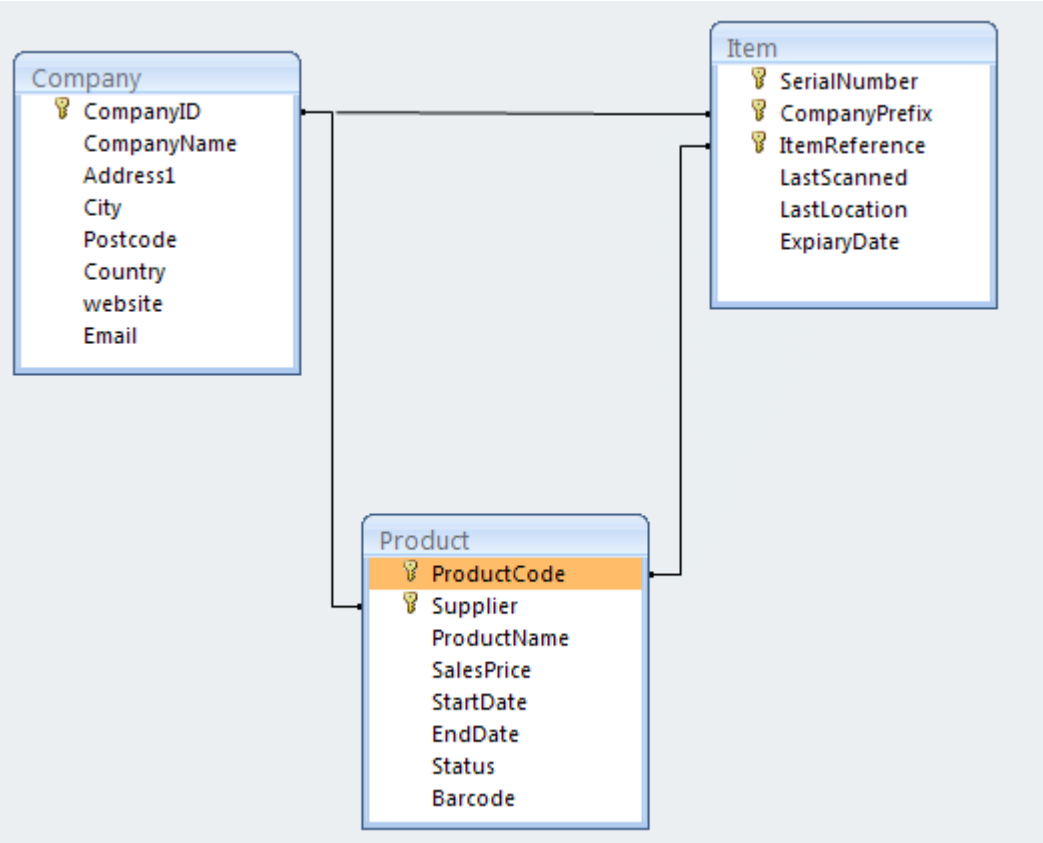
Logical Segment	EPC Header	Filter	Partition	GS1 Company Prefix (*)	Indicator (**) / Item Reference	Serial
Logical Segment Bit Count	8	3	3	20-40	24-4	38

Kuva 5. SGTIN-96 koodaustaulukko (EPC Tag Data Standard, 2011, 96)

Otsakenumero kertoo mitä GS1-yksilöllistä avainta tunniste käyttää. Suodatinnumeroa voidaan käyttää, kun halutaan nähdä vain tietyn tyyppisiä tuotteita. Ositusnumeron avulla saadaan selville yritys ja tuotenumeron pituus. Yritysnumero ja tuotenumero kertovat, mikä tuote on kyseessä ja kuka sen on valmistanut. Sarjanumerolla voidaan antaa jokaiselle tuotteelle oma yksilöllinen numero.

3.3.2 Tietokanta

Ohjelma käyttää Access-tietokantaa, joka koostuu kolmesta taulukosta. Ensimmäisessä taulukossa on yritystiedot, toisessa taulukossa on tuotetiedot ja kolmas käsittää tunnisteet.

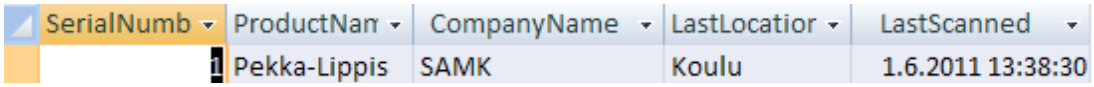


Kuva 6. Tietokannan rakenne ja relaatiot.

Viivakoodeista poiketen tunnisteet sisältävät yritystunnisteen ja tuotekoodin lisäksi myös sarjanumeron. Tuotetaulukon listatessa kaikkien yritysten tuotteet tarvitaan kaksi perusavainta. Tämä siksi, koska kahdella tuotteella voi olla sama tuotenumero mutta eri valmistaja. Samasta syystä tunnistetaulukko vaatii kolme perusavainta. Käytännössä tunnistetaulukon olisi voinut toteuttaa myös yhdellä perusavaimella käyttäen perusavaimena sähköistä tuotekoodia. Sähköisen tuotekoodin heksadesimaalisesta muodosta johtuen, tämä vaihtoehto ei olisi yhtä joustava.

3.3.3 Tulostettavat tiedot

Ohjelma muuttaa sähköisen tuotekoodin yritystunnisteeksi ja tuotenumeroksi. Näillä ohjelma hakee tietokannasta yrityksen nimen ja mikä yrityksen monista tuotteista on kyseessä. Samalla ohjelma muokkaa tietokantaan lukemansa tunnisteen nykyisen sijainnin ja lukuhetken aikaleiman vastaamaan tunnisteen nykyistä sijaintia.



Kuva 7. Käyttäjälle näkyvät tiedot.

3.3.4 Ohjelman perusasetukset

Ohjelmiston perusasetukset tallennetaan erilliseen asetustiedostoon. Tähän tiedostoon tallennetaan mm. lukijalaitteen ja tietokannan osoite. Tulevaisuudessa tähän asetustiedostoon voidaan tallentaa myös muita käyttäjäkohtaisia asetuksia.

4 TUOTANTOPROSESSI

Saavuttaessamme yhteisymmärryksen siitä minkälainen ohjelma toteutetaan, päätin tehdä lisätutkimusta miten ohjelman suunnittelua ja toteutusta kannattaa lähestyä. Koska ohjelmasta tulee suurin ohjelmointiprojekti, jonka parissa olen työskennellyt, halusin tehdä sen kunnolla. Aloitin tämän tutustumalla ohjelmistotuotannon kirjallisuuteen sekä palauttamalla mieleen ohjelmistotuotannon oppitunneilla käytyjä asioita.

4.1 Elinkaarimalli: Prototyyppimalli

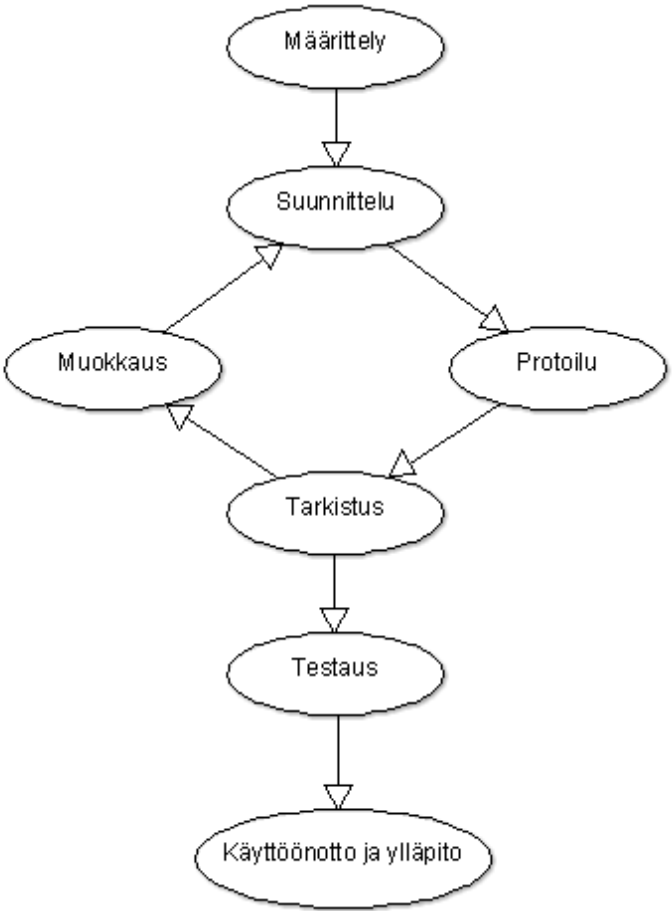
Ohjelmiston elinkaari alkaa ohjelmiston kehitysvaiheen aloittamisesta ja päättyy siihen, kun ohjelmistoa ei enää käytetä, jolloin se poistetaan käytöstä. Ohjelmiston elinkaarimallit kuvaavat miten kyseisen ohjelmiston koko elinkaari on jaettu eri vaiheisiin. (Haikala & Märijärvi, 2004, 36)

Vaikka vanha toteamus ”Hyvin suunniteltu on puoliksi tehty” pätee erinomaisesti ohjelmointiin jättää toteamus kertomatta, että hyvän suunnitelman laatiminen vaatii paljon ammattitaitoa. Kokemattomuuteni vuoksi otin varman päälle ja päätin soveltaa projektissa prototyyppimenetelmää.

Prototyyppimenetelmässä ohjelmasta luodaan prototyyppi, jonka avulla ohjelman käyttöliittymää ja toiminnallisuuksia voidaan kokeilla ennen varsinaisen ohjelman kehittämistä. Menetelmässä ohjelman käyttöliittymä rakennetaan ennen varsinaista toiminnallisuutta. Käyttöliittymää muokataan niin kauan kunnes asiakas on siihen tyytyväinen. Tämän jälkeen käyttöliittymään lisätään toiminnallisuuksia, kunnes prototyyppi vastaa käyttäjän tarpeita. Prototyypin valmistuessa luodaan varsinainen ohjelma sen perusteella.

Projektissa käyttämäni prototyyppimenetelmä alkaa suunnittelusta, jossa tehdään prototyypin alustavat suunnitelmat. Tämän jälkeen siirrytään prototyypin luontivaiheeseen missä prototyyppi luodaan. Prototyypin luomisen tai toiminnallisuuden li-

säyksen jälkeen tarkistetaan ja arvioidaan prototyyppiä ja tehdään siihen tarvittavat muokkaukset. (Haikala & Märijärvi, 2004, 42)



Kuva 8. Prototyyppimenettelyn elinkaarimalli

Menetelmä sopii mielestäni ohjelmointiprojektiini hyvin, sillä prototyyppien avulla on mahdollista kokeilla suunnittelussa tehtyjä ratkaisuja. Mikäli jokin ratkaisu osoittautuu huonoiksi, on se vielä muokattavissa. Prototyyppejä voi myös esitellä asiakkaille ja saada sen perusteella arvokasta palautetta. Menetelmän huonona puolena on prototyyppien tuoma työkuorma. Ohjelmaan tehtävät muutokset voivat osoittautua jopa niin suuriksi, että uuden prototyypin luominen on helpompaa kuin vanhan korjaaminen. Tämä ei silti tarkoita sitä, etteikö vanhan ohjelmakoodin toimivaa osuutta voisi hyödyntää.

5 SUUNNITTELU

Ohjelmistosuunnittelun tarkoituksena on vastata kysymykseen: ”miten ohjelmisto toteutetaan?” Ohjelmistosuunnittelussa kuvataan siis järjestelmää ja sen osia teknillisemmin ja yksityiskohtaisemmin. Perinteisestä vesiputousmallista poiketen prototyyppimenettelyssä ohjelmistosuunnitelmaan tulee prototyyppi vaiheessa vielä paljon muutoksia. Varsinainen ohjelmistosuunnitelma lyödään lukkoon, kun asiakas on tyytyväinen prototyypin toimintaan. (Haikala & Märijärvi, 2004, 81)

5.1 Käyttöliittymä suunnittelu

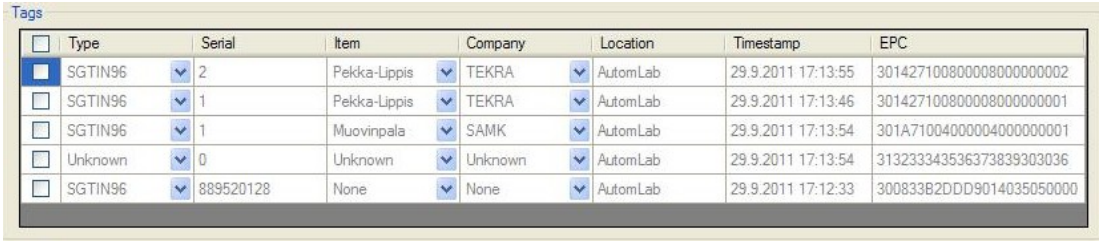
Aloitin ohjelman suunnittelun prototyyppimenetelmän mukaisesti käyttöliittymästä. Helppokäyttöisen käyttöliittymän suunnittelu ei sikäli osoittautunut projektissa ongelmaksi, sillä ohjelman päätarkoituksena oli lähinnä antaa käyttäjälle lista havaituista tunnisteista ja antaa hänelle työkalut näiden tietojen muokkaamiseen. Käyttöliittymään kokonaisuudessaan voi tutustua liitteistä.

5.1.1 Taulukko

Mikäli käyttöliittymässä halutaan listata suuri määrä tietoa, on tähän paras ratkaisu yleensä taulukko. Taulukoiden parhaisiin puoliin kuuluu tietysti, että tiedon voi yleensä järjestää haluamallaan tavalla, ja niiden kautta onnistuu myös tietojen vaivaton muokkaaminen.

Ohjelman kannalta tärkeimmiksi tiedoiksi valitsin:

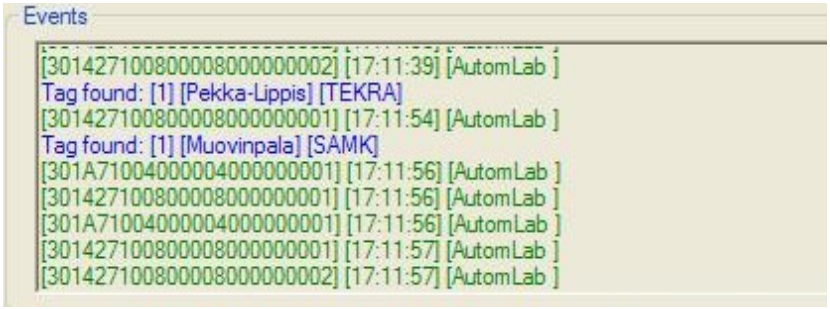
- G1 yksilöinnin avain
- Sarjanumero
- Tuotteen nimi
- Valmistajan nimi
- Sijainti
- Aikaleima
- Sähköinen tuotekoodi



Kuva 9. Tunnistetaulukko toiminnassa.

5.1.2 Tapahtumalaatikko

Ohjelman suorituksen aikana, erityisesti tunnisteiden lukutapahtumassa sattuu ja tapahtuu. Tämän vuoksi katsoin tarpeelliseksi luoda erillisen tapahtumalaatikon johon ohjelma lukijalaitteen luku- ja kirjoitustapahtumista sekä ongelmista. Esimerkiksi lukuhetkellä ympäristöstä ei välttämättä löydy lainkaan tunnisteita, joten ohjelma laittaa lukijalaitteelta tulleen virheilmoituksen tapahtumalaatikkoon. Tapahtumalaatikon perusteella käyttäjälle on myös, selvää milloin ohjelma suorittaa taustalla jotain toimintoa kuten lukee tunnisteita tai kirjoittaa tunnisteisiin.



Kuva 10. Tapahtumalaatikko toiminnassa

5.2 Arkkitehtuurisuunnittelu

Arkkitehtuurisuunnittelussa ohjelmalle suunnitellaan sen sisäinen rakenne. Ohjelmisto jaetaan useisiin mahdollisimman vähän toisistaan riippumattomiin osiin eli moduuleihin. Näin ohjelmaa ei enää tarvitse tarkastella yhtenä suurena kokonaisuutena, vaan ohjelmaa voi ajatella kokoelmana pienempiä toiminnallisia kokonaisuuksia joita voi olla esimerkiksi käyttöliittymä tai tietokanta. Arkkitehtuurisuunnittelussa määritellään myös näiden moduulien väliset rajapinnat. Onnistuneella moduulijaolla ja rajapintojen suunnittelulla voidaan selkeyttää ohjelmiston rakennetta merkittävästi,

vähentää moduuleihin kohdistuvien muutosten vaikutusta muihin moduuleihin sekä parantaa ohjelmakoodin uudelleenkäytettävyyttä. (Koskimies & Mäkinen, 2005, 18)

5.2.1 Moduulijako

Käyttöliittymän suunnittelemisen jälkeen ohjelmiston varsinaisen moduulijaon selvittäminen ei ollut kovin vaikeaa. Koska ohjelman päätoiminto on lukijalaitteen avulla tunnistaa ja hakea lisätietoja tunnisteista on selvää, että moduuleita ovat ainakin tunnisteet, lukijalaite ja tietokanta.

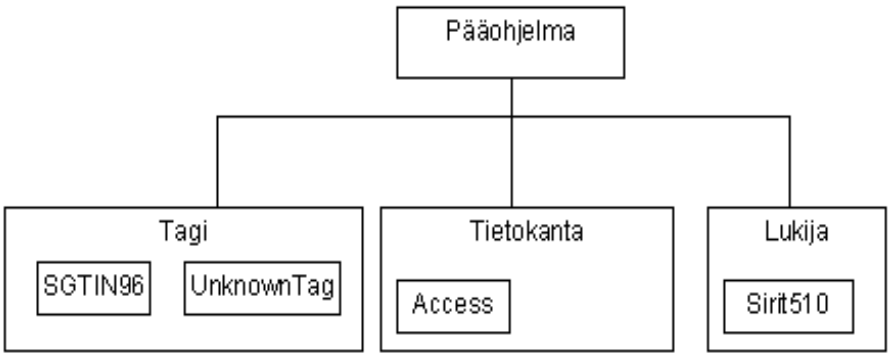
Vaikka käytännössä tiedetään tarkkaan mitä tietokantaa, lukijalaitetta tai yksilöllistä avaimia ohjelma tulee tukemaan on paras hyödyntää moduulien kohdalla abstraktimpaa ajattelumallia, sillä moduulit tulevat keskustelemaan keskenään rajapintojen kautta. Näin tuen lisääminen uusille lukijalaitteille, yksikölillisille avaimille sekä tietokannoille helpottuu merkittävästi.

Moduulijako

- Tunnisteet
- Lukijat
- Tietokanta
- Käyttöliittymä

5.2.2 Alkuperäinen arkkitehtuuri

Myönnettäköön etten heti alkuvaiheessa jaksanut perehtyä ohjelmistoarkkitehtuureihin juuri lainkaan. Ensimmäisten prototyyppien arkkitehtuurin huonot ratkaisut ja puutteet osoittautuivat hyvin opettavaisiksi.

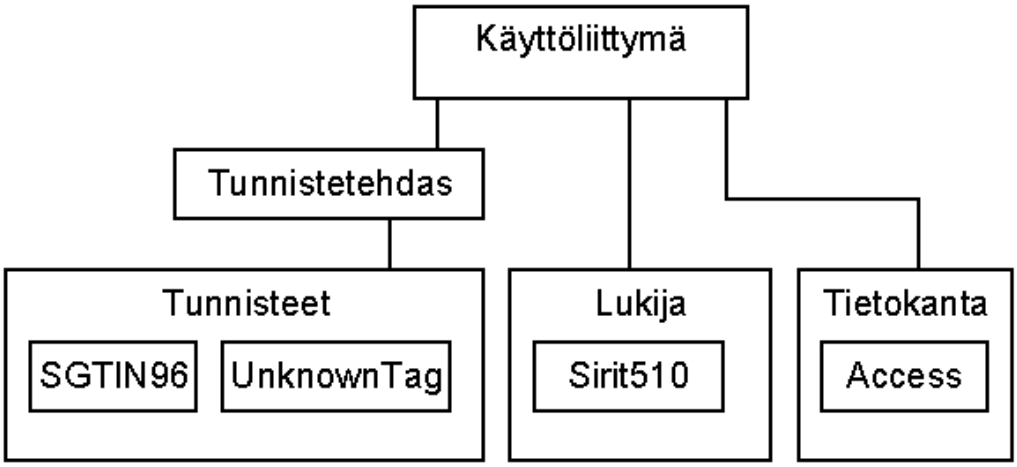


Kuva 11. Alkuperäinen arkkitehtuuri

Alkuperäisessä arkkitehtuurissa ohjelma koostui pääohjelmasta, tunnisteista, tietokannasta ja lukijalaitteesta. Tunnisteille ja lukijalaitteelle löytyi tietysti rajapinnat joilla ohjelmakoodin kautta oli mahdollista käyttää useita eri GS1-yksilöinnin avaimia, sekä lukijalaitteita. Arkkitehtuurin ongelmaksi muodostui pääohjelma ajattelumalli, eli ohjelmakoodi jakautui moduulien välille hyvin epätasapainoisesti. Pääohjelma vastasi tunniste- ja lukijaolioiden luomisesta sekä tietokannan käytöstä. Ohjelmakoodi muotoutui hetki hetkeltä sekavammaksi ja ohjelman rakenne alkoi ennepitkään natista liitoksissaan.

5.2.3 Toinen arkkitehtuuri

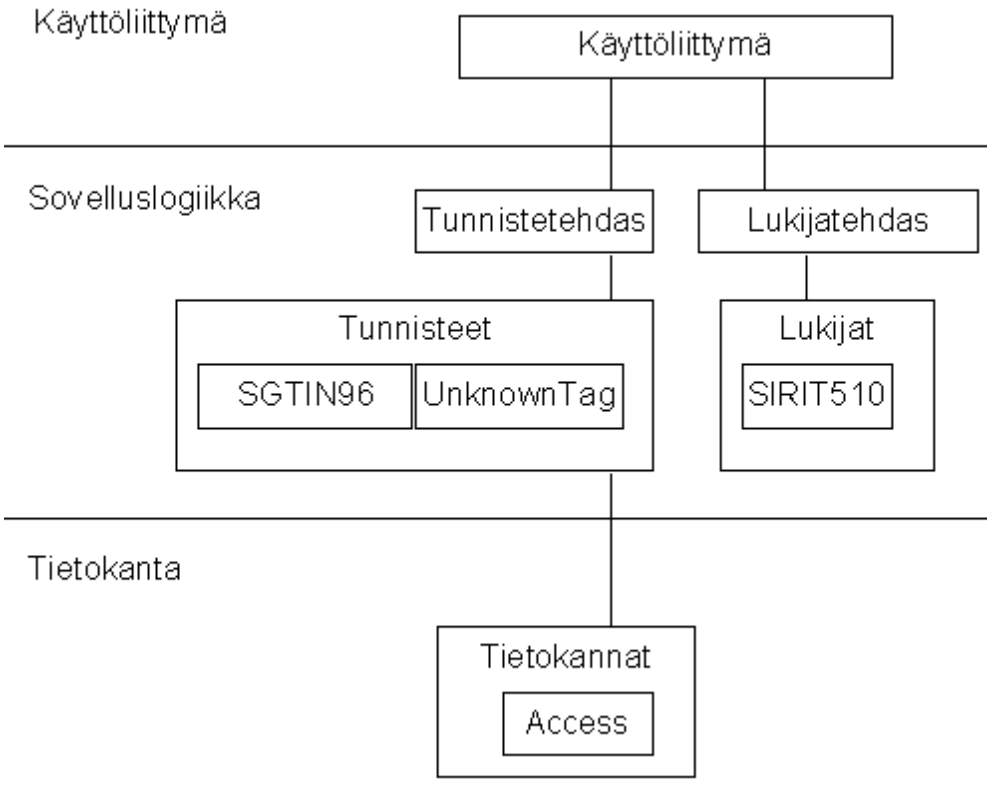
Toisessa arkkitehtuurissa hylkäsin pääohjelma ajattelun ja ajattelin ohjelman ylintä osaa käyttöliittymänä. Käyttöliittymä vastasi edelleen lukijalaitteen sekä tietokannan käytöstä. Tunnisteiden luonnissa hyödynsin menetelmää, joka myöhemmin valkeni tehdas -suunnittelumalliksi. Ensimmäiseen verrattuna arkkitehtuuri toimi huomattavasti paremmin. Valitettavasti tietokannan hyödyntäminen käyttöliittymä moduulin kautta osoittautui todella kömpelöksi.



Kuva 12. Toinen arkkitehtuuri.

5.2.4 Lopullinen arkkitehtuuri

Päätin tutustua arkkitehtuureihin hieman paremmin ja törmäsin kerrosarkkitehtuuriin. Kerrosarkkitehtuurissa ylemmän tason osat käyttävät hyväkseen alemman kerroksen palveluita. Kolmen kerroksen arkkitehtuurissa alin kerros eli tietokantakerros vastaa tiedon säilytyksestä. Kerroksen yläpuolella on tietokantaa hyödyntävä sovellukseen liittyvästä logiikasta vastaava kerros. Ylimmästä kerroksesta löytyy tietysti käyttöliittymä, joka vastaa sovelluslogiikkakerroksesta pyytämänsä- ja saamansa tiedon esittämisestä. (Koskimies & Mäkinen, 2005, 126–127)



Kuva 13. Lopullinen Arkkitehtuuri

Lopullisessa arkkitehtuurissa sain viimein ratkaistua aiemmissa arkkitehtuureissa esiintyneet ongelmat. Käyttöliittymään liittyvä koodi pysyy siistinä, sillä tunnisteisiin ja lukijalaitteisiin liittyvä sovelluskohtainen logiikka toteutetaan näiden omissa luokissa. Tunnisteet hakevat tietokannasta tarvitsemansa tiedot kuten yritysnumeroa vastaavan nimen tietokannasta luontihetkellä. Kerrosarkkitehtuurin myötä ohjelman kehityksestä tuli huomattavasti selkeämpää ja organisoidumpaa.

5.3 Suunnittelumalli

Suunnittelumalleilla tarkoitetaan jotain uudelleenkäytettäviä menetelmiä tai malliratkaisuja jonkin yleisen ongelman tai tilanteen ratkaisemiseksi. Nämä menetelmät on yleensä todettu kokemuksen ja testien myötä toimiviksi useissa eri tilanteissa. Koska suunnittelumalleissa kerrotaan usein tarkasti miksi mallin mukaisesti menetellään, katsoin suunnittelumallit erinomaiseksi apuvälineeksi oliopohjaisten suunnittelu ratkaisujen opetteluun.

5.3.1 Tehdasmetodi

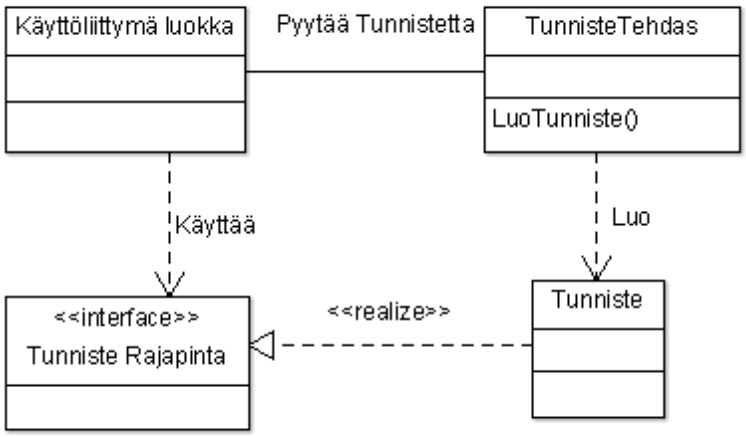
Kuten lopullisesta arkkitehtuurista huomaa, päätin soveltaa työssäni tehdasmetodi suunnittelumallia. Syynä tähän on ohjelman osien monimuotoisuus.

Taulukko 2. Monimuotoisuus

RFID-lukijalaitteet	Useita valmistajia ja malleja.
Tunnisteet	GS1-yksilölliset avaimet ja tuntemattomat tunnisteet.
Tietokannat	Esimerkiksi Access, MySQL ja Oracle.

Tehdasmenetelmän avulla tunnisteet ja lukijalaitteet luodaan erillisessä tehdasluokassa. Kaikki lukija- ja tunnisteluokat toteuttavat lukijarajapinnan määrittämät toiminnot. Tehtaan luomia olioita käytetään rajapintojen avulla, näin ainoastaan tehdas on tietoinen luokkien tarkasta rakenteesta. Tämä helpottaa merkittävästi uusien lukijalaitteiden ja tunnisteiden lisäämistä ohjelmakoodiin, sillä esimerkiksi uusia lukijalaiteluokkia lisätessä muutoksia tulee lähinnä lukijatehdasluokkaan.

Kuvasta 14. näkee miten tunnistetehdas toimii. Käyttöliittymä pyytää tunnistetehtaalta tunnisteoliota. Tehdas antaa luomansa tunnisteolion käyttöliittymälle, joka käyttää sitä tunnisterajapinnan kautta.



Kuva 14. Tunnistetehdas

6 TOTEUTUS

6.1 Välineet

6.1.1 Visual Studio -kehitysympäristö

Ohjelma toteutuksessa käytin Microsoftin Visual Studio -kehitysympäristöä. Ympäristö tukee useita oliopohjaisia ohjelmointikieliä millä ympäristössä voi kehittää esimerkiksi Windows- ja mobiilisovelluksia sekä websivuja. Päätin toteuttaa ohjelman tässä ympäristössä, sillä se on tullut minulle tutuksi opintojen myötä.

6.1.2 C-sharp(C#)

Ohjelman toteutuskieleksi valitsin Microsoftin kehittämän oliopohjaisen C# ohjelmointikielen. Ohjelmointikieli ei sikäli ollut aloittaessa kovinkaan tuttu, mutta siihen siirtyminen Vb.Net-ohjelmointikielestä osoittautui vaivattomaksi. Kieli on suunniteltu erityisesti .Net -ohjelmistokomponenttikirjaston hyödyntämiseen. Kielen kehityksessä tavoitteena on ollut yhdistää C++- ja Java-kielen parhaat puolet. (Wikipedia www-sivut)

6.1.3 Microsoft Access

Access on Microsoftin lanseeraaman Office-ohjelmistopaketin mukana tuleva tietokantaohjelma. Sen avulla voi helposti luoda ja hallita ohjelmalla tehtyjä tietokantoja. Access-tietokannan käyttö onnistuu Visual Studiossa monella eri tapaa. Tietokannan voisi esimerkiksi liittää osaksi projektia ja toteuttaa kaikki halutut toiminnot graafisen käyttöliittymän kautta. Päätin soveltaa standardoitua kyselykieltä (SQL, Structured Query Language) ja hallita tietokantaa ohjelmakoodin kautta.

6.2 Käytännöt

Koska ohjelman jatkokehityksestä tulee todennäköisesti vastaamaan joko tietotekniikan oppilaat tai mahdollisesti opettajat. Halusin koodin olevan siistiä, organisoitua ja

helposti ymmärrettävissä. Tämän vuoksi päätin soveltaa koodissa yleisiä hyviä ohjelmointitapoja.

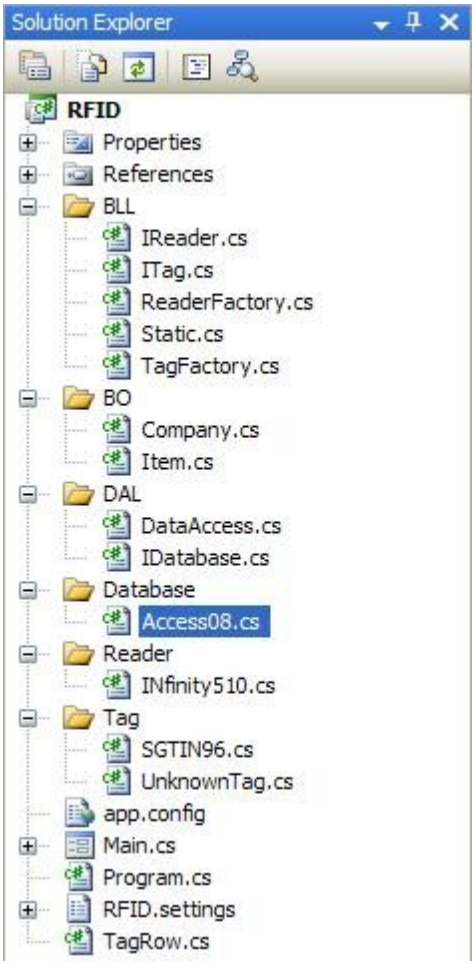
Suurin osa ohjelmointia ja tietotekniikkaa käsittelevästi kirjallisuudesta sekä verkkomateriaalista on englanninkielistä, joten päädyin käyttämään sitä myös ohjelmassa. Käyttöliittymän ja ohjelmakoodin ollessa englannin kielellä myös vaihto-oppilaat voivat käyttää ja mahdollisesti vastata ohjelmiston jatkokehityksestä.

6.2.1 Nimeämiskäytännöt

Nimeämiskäytännöissä sopivimmaksi vaihtoehdoksi osoittautui MSDN:n (Microsoft Developer Network) laatimat nimeämisohjeet(Naming Guidelines). Nimeämisohjeet sisältävät tarkat ohjeet siitä miten muuttujat, luokat, oliot yms. kannattaa nimetä. Tämän nimeämisohjeen käyttö on suositeltavaa erityisesti kehittäessä ohjelmistoja, jotka hyödyntävät .Net -ohjelmistokomponenttikirjastoa. Tällöin kehitettävän ohjelman nimeämiskäytännöt vastaavat Microsoftin vastaavia.

6.2.2 Kansiojaottelu ja nimiavaruudet

Sovellusnäkymässä järjestin luokat ja rajapinnat niitä kuvaaviin kansioihin. Kansioiden myötä myös ohjelmiston nimiavaruuden hyödyntäminen tehostuu, sillä jokainen kansioon luotu luokka saa automaattisesti nimiavaruuden kansion nimen perusteella. Esimerkiksi SGTIN96-luokkaa voi käyttää vain ne ohjelman osat jotka käyttävät RFID.Tag -nimiavaruutta. Tällä menetelmällä onnistuu myös kätevästi kerrosarkkitehtuurinmukainen rajaus, jossa käyttöliittymä pääsee käsiksi ainoastaan loogisen kerroksen luokkiin sekä rajapintoihin, loogisen kerroksen päästessä taas käsiksi tietokanta kerrokseen.



Kuva 15. Kansiojaottelu

6.2.3 Kommentointi

C#-ohjelmointikielessä löytyy mekanismi, jonka avulla kehittäjät voivat kätevästi kommentoida koodiaan käyttämällä XML-kommentointia. Kommentointi helpottaa esimerkiksi koodiin liittyvien dokumentointien tekoa. Kehitysympäristö osaa myös itse paremmin hyödyntää XML-muodossa olevia kommentteja eri tarkoituksiin.

```
/// <summary>
/// Funktio joka palauttaa sille parametrina annetun nimen
/// </summary>
/// <param name="Nimi">Nimi merkkojonoa</param>
/// <returns>Nimen merkkijonona</returns>

Funktio("Matti Meikäläinen");
```

string Form1.Funktio(string Nimi)
Funktio joka palauttaa sille parametrina annetun nimen

Kuva 16. XML-dokumentointi

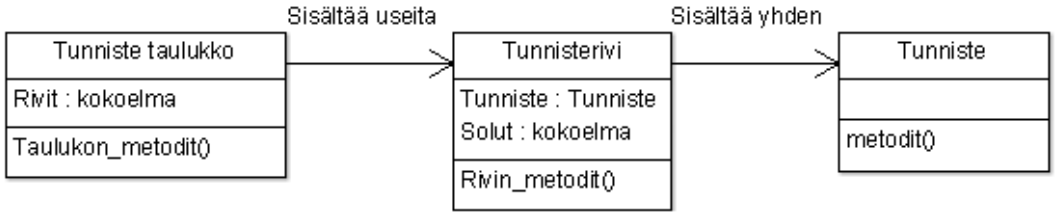
6.3 Ohjelman toteutus

6.3.1 Tunnistetaulukko

Ohjelmakoodissa TagGrid -nimellä kulkevan tunnistetaulukko on yksi ohjelman käyttöliittymäkomponenteista. TagGrid käyttää DataGridView nimistä Windows-lomakkeiden käyttöliittymäkomponenttiluokkaa. Ohjelma listaa tähän TagGrid-taulukkoon kaikki tunnisteet tietoineen joita se havaitsee lukijalaitteella. Myös tunnistetietojen muokkaaminen tapahtuu juurikin tämän tunnistetaulukon kautta.

Varhaisissa prototyypeissä ohjelma loi tunnisteet ja lisäsi ne erilliseen tunnistekokoelmaan. Tunnistekokoelmaan lisätyt tunnisteet ilmestyivät heti TagGrid-taulukkoon, sillä kyseinen komponentti käytti tunnistekokoelmaa omana tietolähteenään. Jokaiselle tunnisteelle sai kätevästi oma rivinsä, josta tietoja pääsi kätevästi muokkaamaan.

Tämä osoittautui ongelmalliseksi kahdesta syystä. Pudotuslistojen käyttö osoittautui tällä menettelyllä erittäin kömpelöksi. Tällä menettelyllä en päässyt haluamallani tavalla vaikuttamaan rivin luontiin tai toiminnallisuuteen. Pitkän pohdinnan jälkeen aloin tutkimaan DataGridView-komponentin sisäisiä luokkia. Huomasin, että kyseinen komponentti tallentaa rivit omaan Rows-kokoelmaansa DataGridViewRow -muodossa.



Kuva 17. Tunnistetaulukko

Palautin mieleeni olio-ohjelmoinnin perusteissa läpikäydyn periytymisen. Olio-ohjelmoinnissa periytymisellä tarkoitetaan sitä tilannetta, jossa jonkin luokan perusteella

luodaan aliluokka, joka perii kaikki isäluokan ominaisuudet. Aliluokalle voi lisätä sellaisia ominaisuuksia mitä isäluokasta ei löydy. Loin siis DataGridViewRow-luokan pohjalta uuden luokan, joka kantaa nimeä TagRow eli suomeksi tunnisterivi. Tunnisterivi on siis täysin tunnisteiden tietojen säilytykseen ja muokkaukseen räätälöity taulukkoriviluokka. Tunnisteriviluokkaa hyödyntämällä taulukon rivien hallinnasta tuli naurettavan helppoa. Esimerkiksi, jos käyttäjä ei haluakkaan tehdä tunnisteisiin muutoksia on todella helppoa tehdä metodi, joka käskee jokaista taulukon riviä palauttamaan alkuperäiset tunnisteen arvot.

6.3.2 Tunnisteoliot

Luokkaa voi ajatella muottina olioille eli luokan ilmentymille, esimerkiksi ohjelman SGTIN96-luokan avulla tehdään kaikista lukijalaitteen havaitsemista EPC SGTIN-96 tunnisteista niitä vastaavat oliot. Näin ohjelma käsittelee jokaista havaittua tunnistetta yksilönä. Tunnisteluokassa määritellään mitä tietojäseniä(Members) ja metodeja(Methods) tunnisteolioista löytyy. SGTIN96-luokan olion tietojäseniä on esimerkiksi tunnistenumero(tagID), Yritystunniste(companyPrefix), sekä tuotenumero(productID). Esimerkkinä luokan metodeista mainittakoon koodaa ja tulkitse metodit joiden avulla tunnisteet osaavat muuttaa EPC-koodin ihmisen ymmärtämään muotoon ja takaisin.

Tunnisteluokat tukevat tunnisterajapintaa(ITag) jota esimerkiksi käyttöliittymä käyttää tunnistetietojen esittämiseen taulukossa. Tunnisterajapinnassa määritellään esimerkiksi tunnisteluokan tarvitsemat ominaisuudet(Properties), metodit sekä funktiot mitä käyttöliittymä tarvitsee tunnistetietojen esittämiseen ja muokkaamiseen. Tunnisterajapinnan myötä käyttöliittymä tietää varmasti, että jokaisesta rajapintaa hyödyntävästä tunnisteluokasta löytyy rajapinnan määrittämät toiminnallisuudet. Esimerkiksi käyttöliittymän kysyessä tunnisteelta tuotteen nimeä, SGTIN96-tunnisteoliot vastaavat tähän tuotteen nimellä ja tuntemattomiksi luokitellut(UnknownTag) tunnisteoliot vastaavat sanalla ”None”. Koska molemmat oliot vastaavat tähän rajapinnan mukaisesti merkkijonolla, käyttöliittymän ei tarvitse tietää miten ne päätyivät vastaukseensa.

6.3.3 SGTIN-96 -tunnisteluokka

SGTIN-96 kooditaulukko on määritelty tarkasti EPCglobalin tunnisteiden tietosisältöjä käsittelevässä standardissa. SGTIN-96 -tunnisteluokka toimii tietysti tämän kooditaulukon mukaisesti.

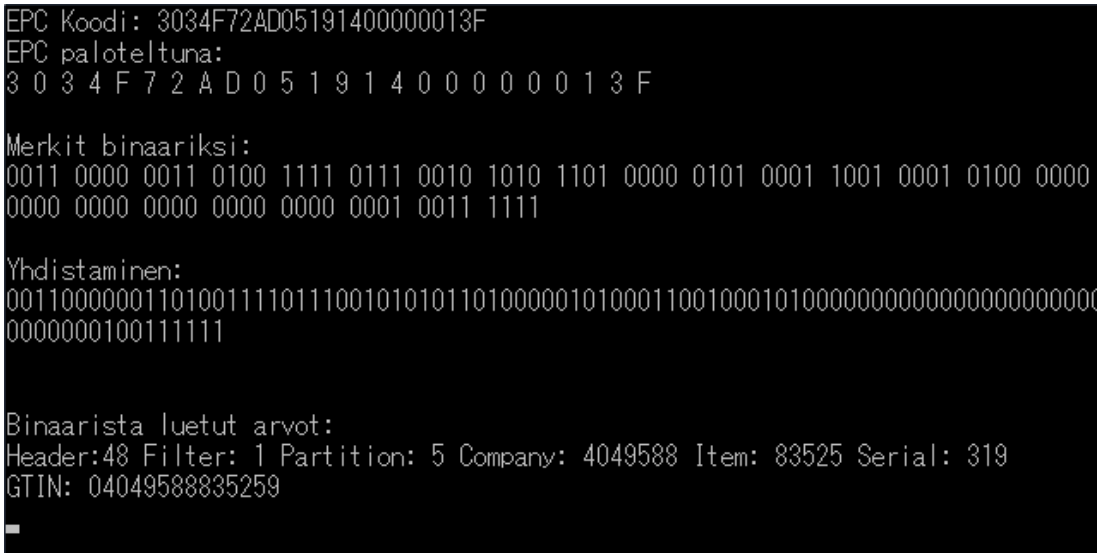
SGTIN-96 -koodaustaulukossa sähköinen tuotekoodi jaetaan kuuteen osaan:

- Header (Otsikko) (8 bittiä)
- Filter (Suodatusnumero)(3 bittiä)
- Partition(Ositusnumero)(3 bittiä)
- Company Prefix(Yritystunniste)(20 – 40 bittiä)
- Item Reference(Tuotetunniste)(24 – 4 bittiä)
- Serial number(Sarjanumero)(38 bittiä)

(EPC Tag Data Standard, 2011, 96)

Käyttöliittymä pyytää tunnistetehtaalta uutta tunnisteoliota antamalla tälle lukijaoliolla lukemansa tunnistenumeron. Tunnistetehdas katsoo tunnistenumeron perusteella minkälaisen tunnisteen se oikein luo. Otsikkoarvon ollessa tasan 8 bittiä pitkä ei sen tulkitseminen heksadesimaalina esitetystä tunnistenumerosta vaadi erillistä binäärimuunnosta. Tehtaan tulkitessa otsikkoarvon perusteella, että kyseessä on SGTIN-96 luo se koodausta vastaavan tunnisteolion.

Tunnisteoliota luodessa heksadesimaalinen tunnistenumero muutetaan binäärimuotoon merkki kerrallaan, kunnes saadaan 96 bittiä pitkä binääriluku. Tunnistenumeroa ei siis käsitellä sähköisessä tuotekoodissa yhtenä suurena heksadesimaalilukuna.



Kuva 18. 96-bittisen sähköisen tuotekoodi SGTIN muotoon.

Kuten kuva 18. kertoo. Sähköistä tuotekoodia kääntäessä jokainen heksadesimaalin merkki muutetaan yksi kerrallaan 4 merkkiä pitkäksi binääriluvuksi. Tämän jälkeen kaikki 24 binäärilukua yhdistetään yhdeksi suureksi 96-bittiseksi binääriluvuksi. Binääriluku luetaan ensimmäisen 8-bitin määrittämän GS1-yksilöinnin avaimen mukaan.

6.3.4 Dynaaminen pudotusvalikko

Kuten aiemmin mainitsinkin, käyttäjä muokkaa tunnistetietoja tunnistetaulukon kautta. Osa taulukon rivien soluista on yleisestä tekstilaatikosta poiketen käyttöliittymäkomponenttiluokaltaan pudotusvalikkoja. Käyttäjän halutessa vaihtaa tunnisteeseen liitettyä valmistajaa ja/tai tuotetta, valittavina on vain tietokannasta löytyviä valmistajia ja heidän tuotteitaan. Tämä helpottaa merkittävästi syötteiden tarkistusta ja tekee asioista myös käyttäjän kannalta mukavampaa.

Ohjelman tunnisterivin solut toimivat siis dynaamisesti. Tunnistetyypin ollessa tuntematon on kaikki tunnisterivin solut lukittuna. Käyttäjän vaihtaessa tyypiksi SGTIN96, hakee ohjelma tietokannasta valmiiksi ensimmäisen yrityksen ja heidän ensimmäisen tuotteensa. Tuotteiden ollessa valmistajakohtaisia käyttäjän vaihtaessa valmistajasolun arvoa, hakee ohjelma valitun yrityksen ensimmäisen tuotteen ja täyttää tuotenimi pudotusvalikon yrityksen tuotteilla.

6.3.5 Säikeet

Ohjelmassa tunnisteiden luku tapahtuu taustalla pyörivän säikeen avulla. Tämä säie luodaan luentaa aloittaessa ja lopetetaan ohjelma lopettaessa. Säie kutsuu havaitessaan uuden tunnisteen pääsäiettä lisäämään tunnisteen tunnistetaulukkoon. Säie seuraa ohjelman sisäistä lukutilaa. Lukutilan saadessa arvon Epätosi säie pääsee ulos silmukasta ja häviää. Tilaa muutetaan Aloita luenta, Lopeta luenta -painikkeilla, mutta myös ohjelman sammuttaminen lopettaa luennan. Näin säikeiden pysäyttäminen toimii ”pehmeästi” eikä aiheuta virheitä.

6.3.6 Lukijalaiteluokka

Kirjoitushetkellä koululla on käytössä vain yksi RFID-lukijalaite. Ohjelman lukijalaiteluokka kantaa koulun lukijalaitteen mallin mukaisesti nimeä Sirit510. Se käyttää lukijalaitteen ohjelmistojen mukana tullutta laitteen käyttöön tarkoitettua kirjastoa (Rapid.dll), jonka kautta lukijalaitteen säätäminen ja käyttäminen on melko vaivatonta. En katsonut tarpeelliseksi käyttää kirjastoa muihin tarkoituksiin kuin tunnisteiden lukemiseen sekä niihin kirjoittamiseen.

Sirit tarjoaa kattavat ohjeet lukijalaitteiden käyttöön lukijalaitteen tukisivulla:
http://www.sirit.com/INfinity_510_Support.asp

Tunnisteen lukeminen tapahtuu lähettämällä lukijalaitteelle read_id -komennon, tällöin lukijalaite lukee ensimmäisen havaitsemansa tunnisteen tunnistenumeron ja palauttaa sen sitä käyttävälle ohjelmalle. Kirjoitus tapahtuu lähettämällä tunnisteelle write_id() -komento, jonka sulkujen sisään laitetaan uusi ID sekä sen tunnisteen ID johon halutaan kirjoittaa.

7 POHDINTA

Työn tavoitteena oli toteuttaa koululle opetukseen soveltuva RFID-seuranta- ja tunnistusjärjestelmä. Asiakas antoi minulle täysin vapaat kädet nykyisille RFID-lukijalaitteille ja -tunnisteille suunnatun ohjelman toteutuksessa. Tämän vuoksi olisin voinut monessa kohtaa mennä sieltä, mistä aita on matalin. Otin tämän vastaan enemmänkin haasteena ja mahdollisuutena syventyä tarkemmin RFID-tekniikkaan ja ohjelmistokehityksen maailmaan. Opinnäytetyön kirjallisen osuuden puolesta keskittyminen pelkkään ohjelmointiprojektiin osoittautui varsin haastavaksi. Monista projektin aikana käydyistä aihealueista kuten arkkitehtuurisuunnittelusta ja RFID-tekniikasta voisi helposti kirjoittaa kokonaisen opinnäytetyön. Kirjallisen osuuden suurimmaksi haasteeksi osoittautuikin aihealueen rajaaminen projektin kannalta oleellisimpiin asioihin.

Tämän ohjelmointiprojektin ollessa ylivoimaisesti suurin ohjelmointityö mitä olen tähän päivään mennessä tehnyt, se on monella tapaa ollut hyvin opettavainen. RFID-osuus osoittautui tekniikan ja standardien puolesta varsin haastavaksi. Tämä näkyi erityisesti sähköisen tuotekoodin kohdalla, missä standardi oli täynnä teknistä nippelitietoa ja vaikutti aluksi todella sekavalta. Käsitteisiin tutustuessa palaset loksahtivat pikkuhiljaa paikoilleen. Suunnittelupuolella perehdyttyäni erityisesti arkkitehtuurisuunnitteluun ja suunnittelumalleihin ymmärrykseni olio-ohjelmoinnista ja sitä hyödyntävistä tekniikoista parantui huomattavasti. Lukijalaitteen myötä pääsin myös tutustumaan laitekirjastoihin sekä laiteohjelmointiin, joka osoittautui lukijalaitteen kohdalla odotettua helpommaksi. Prototyyppimallin hyödyntäminen osoittautui oppimisen kannalta erinomaiseksi valinnaksi. Ohjelman prototyypeissä vastaan tulleet virheet ja ongelmat pakottivat kokeilemaan uusia lähestymistapoja sekä opettelemaan uusia ohjelmointiin liittyviä käsitteitä ja asioita.

Miettiessäni, mitä olisin voinut tehdä opinnäytetyössäni paremmin, huomasin, että käytännössä ohjelmistotuotannossa tehdään huomattavasti enemmän dokumentointia kuin pelkkä määrittelydokumentti. Suunnittelumalleja löytyi aika tavalla, joten näistä

olisi varmaan löytynyt useampia ohjelmaan sopivia ratkaisuja. Tietokannan puolesta mainittakoon, että olisin voinut tutustua paremmin Microsoftin tarjoamiin tietokannanhallinta ominaisuuksiin Visual Studio-kehitysympäristössä. Opinnäytetyön kohdalla olisin halunnut lisätä ohjelman testauksesta ja käyttöönotosta vielä kattavat osiot, mutta näiden myötä opinnäytetyö olisi jo paisunut aika suureksi.

LÄHTEET

RFIDLab www-sivut. Viitattu 15.6.2011 http://www.rfidlab.fi/

RFID Journal www-sivut. Viitattu 15.6.2011 http://www.rfidjournal.com/

Glover, B.. & Bhatt, H. *2006. RFID Essentails 2006. USA: O'relly media inc*

SkyRFID. RFID Gen2 – What is It? - Smart RFID! www-sivut Viitattu 29.9.2011 http://www.skyrfid.com/RFID_Gen_2_What_is_it.php

Sarekoski,S. Yleistä tietoa RFID:stä. Www-sivut. Viitattu 29.9.2011. http://www.sareskoski.fi/rfid.htm

UHF Class 1 Gen 2 Standard v. 1.2.0. 2008. GS1 EPCglobal Inc. Viitattu 29.9.2011. http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf

GS1 EPC Tag Data Standard 1.6. 2011. GS1 EPCglobal Inc. Brussels, Belgium. Viitattu 29.9.2011. http://www.gs1.org/gsmp/kc/epcglobal/tds/tds_1_6-RatifiedStd-20110922.pdf

Koskinen, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki: TALENTUM

Haikala, I & Märijärvi, J. 2004 Ohjelmistotuotanto. Helsinki: TALENTUM

Wikipedia www-sivut. Viitattu 30.9.2011. http://fi.wikipedia.org/wiki/C_sharp

Wikipedia www-sivut Viitattu 30.9.2011 http://fi.wikipedia.org/wiki/RFID

Salvén, K. Älyä puuntyöstöprosesseihin. 2011. http://www.kantti.net/artikkeli/2011/03/%C3%A4ly%C3%A4-puunty%C3%B6st%C3%B6prosesseihin-rfidll%C3%A4

Granqvist, J., Permala, A.& Scholliers, J. 2007. RFID-tunnistus logistiikan kehittämisessä. Espoo: Logistiikan Osaamiskeskus. http://www.rfidlab.fi/sites/rfidlab.fi/files/RFTUNLOG_Tutkimusraportti_Final%2012.2.2007.pdf

LIITE 1

14.5.1.1 SGTIN-96 Coding Table

Scheme	SGTIN-96					
URI Template	`urn:epc:tag:sgtin-96:F.C.I.S`					
Total Bits	96					
Logical Segment	EPC Header	Filter	Partition	GS1 Company Prefix (*)	Indicator (**) / Item Reference	Serial
Logical Segment Bit Count	8	3	3	20-40	24-4	38
Coding Segment	EPC Header	Filter	GTIN			Serial
URI portion		*F*	*C.I*			*S*
Coding Segment Bit Count	8	3	47			38
Bit Position	$b_{95}b_{94}\ldots b_{88}$	$b_{87}b_{86}b_{85}$	$b_{84}b_{83}\ldots b_{38}$			$b_{37}b_{36}\ldots b_0$
Coding Method	00110000	Integer	Partition Table 17			Integer

Table 18. SGTIN-96 Coding Table

Partition Value (*P*)	**GS1 Company Prefix**		**Indicator/Pad Digit and Item Reference**	
	Bits (*M*)	**Digits (*L*)**	**Bits (N)**	**Digits**
0	40	12	4	1
1	37	11	7	2
2	34	10	10	3
3	30	9	14	4
4	27	8	17	5
5	24	7	20	6
6	20	6	24	7

Table 17. SGTIN Partition Table

EPC SGTIN-96 koodi- ja ositustaulukko.

LIITE 2

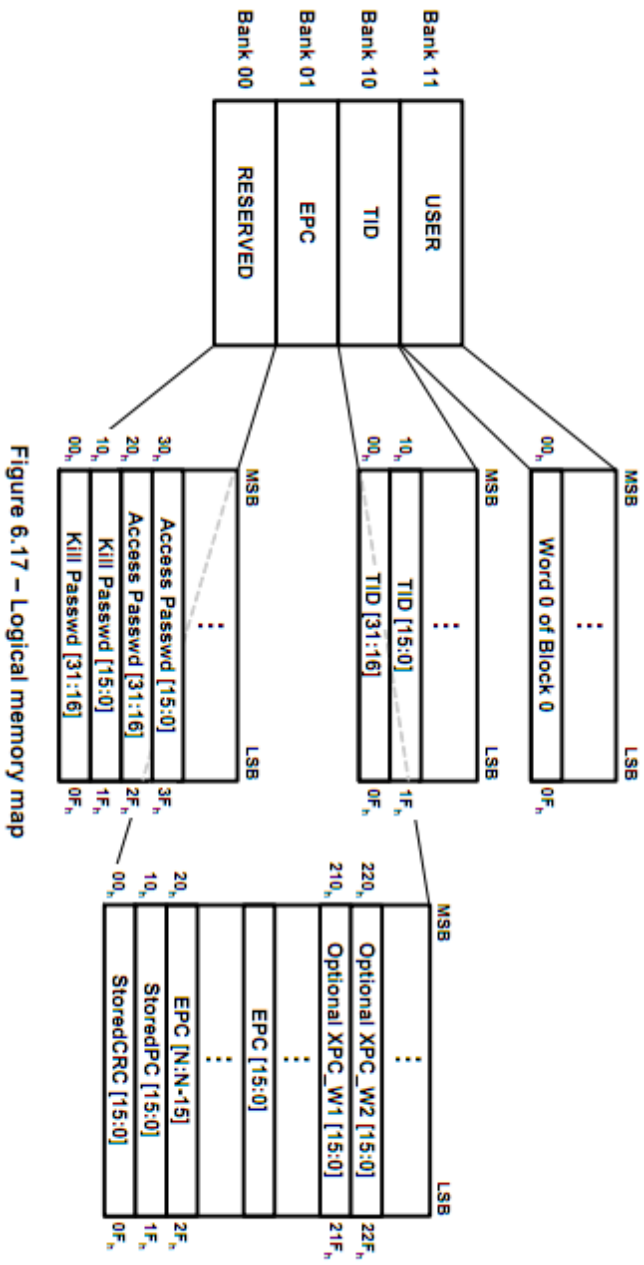
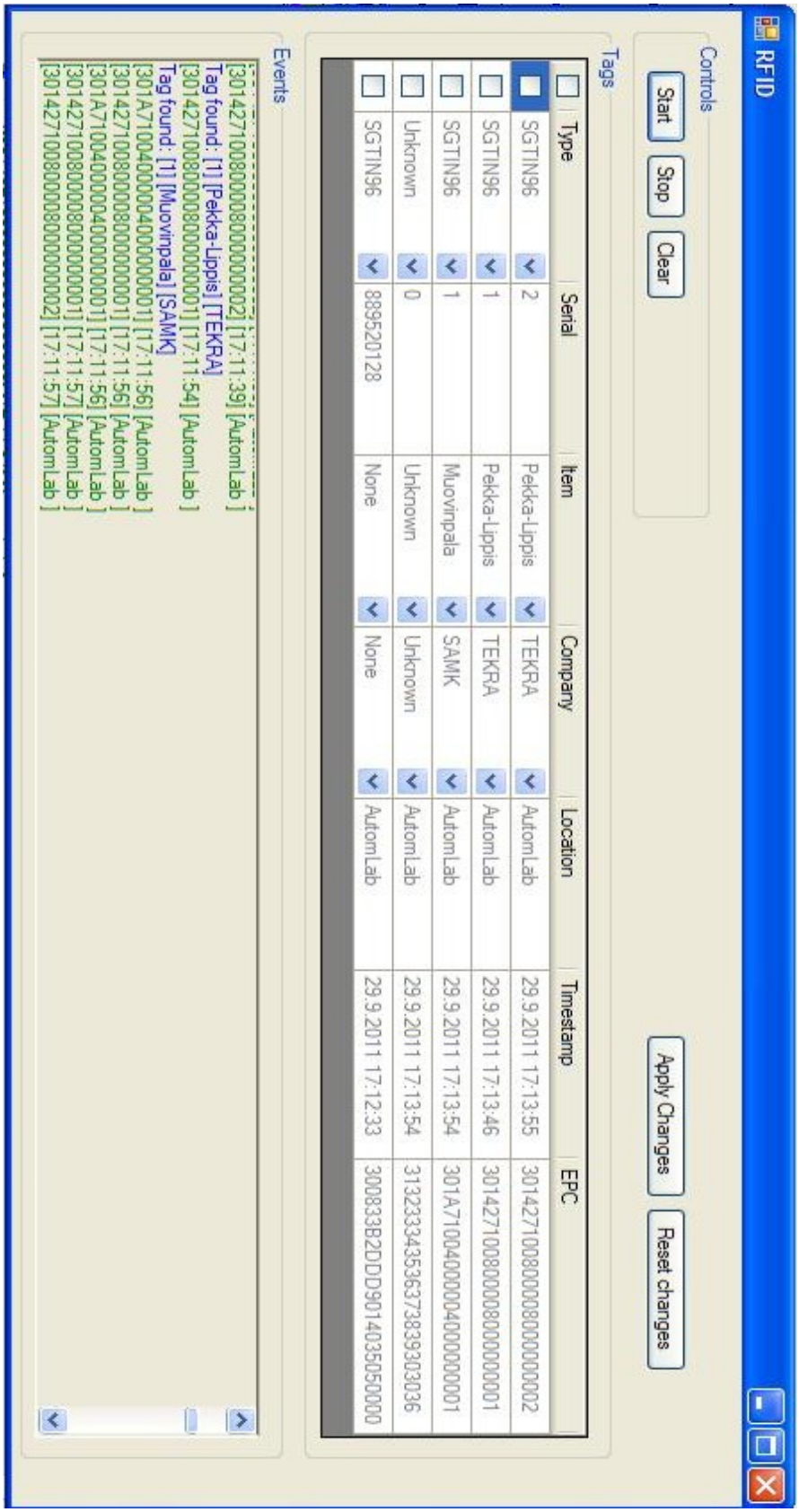


Figure 6.17 – Logical memory map

Gen2-tunnisteen muistilohkojen rakenne.

LIITE 3



Valmis käyttöliittymä